Name :- Dheeraj Soni
Section :- CE
Roll No :- 23

Tutorial- 05

Ques1 Using Bfs, we can find the minimum number of nodes between source node and destination node, while using Dfs, we can find if a path exist between two nodes.

Application of Dfs :-

1) Detecting cycle in a graph.
2) finding path between two given vertices U&V
3) Topological sorting can be done using Dfs.

Application of Bfs :-

1) Like Dfs, Bfs may also be used for detecting cycles in graph.
2) Finding shortest path and minimum spanning tree in unweighted graph.
3) In networking, finding a route for packet transmission.

Ques 2 which data structures are used for implementing Bfs and Dfs & why?

Dfs uses stack data structure as order doesn't has much importance.

Bfs uses queue data structure as order matters in this case.

Ques3 Sparse Graph :- Graph in which no. of edges is much less than the possible no. of edges.

Dense Graph :- Graph in which no. of edges is close to the maximal no. of edges

If the graph is sparse, we should store it as a list of edges. Alternatively, if it is dense, we should store it as adjacency matrix.

Ques 4 Detecting a cycle in graph

Using BFS :-

1) Complete in degree (no. of incoming edges) for each of the vertex present in ~~nodes~~ graph & count no. of nodes = 0
2) pick all vertices with indegree as 0 and add them to queue
3) Remove a vertex from the queue, ~~then~~ ~~increment current count~~ increment count by 1 and decrease in degree by 1. for all neighbours. if in-degree of a neighbouring node is 0 add to queue.
4) Repeat step 3 untill queue is empty
5) If no. of visited nodes is not equal to no. of nodes, then graph has a cycle.

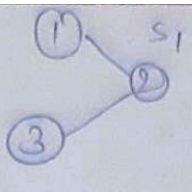
Using DFS :-

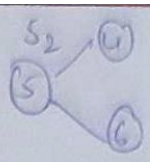
Similar process is done in DFS as well, but in DFS, we has the option of doing recursive called for vertices which are adjacent to the current node & are not get visited. If recursive function returns false, then graph does not have a cycle.

Ques 5 Disjoint Set :- It allows to find out whether the 2 elements are in the same set or not efficiently. The disjoint set can be divided as the subsets where there is no common element between 2 sets.

$S_1 = \{1, 2, 3\}$

$S_2 = \{4, 5, 6\}$

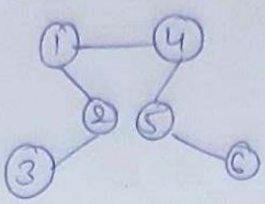




Operations :-

1) Union :- merge 2 sets when edge is added.

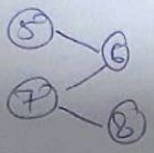
$S_1 \cup S_2 = S_3 \Rightarrow$



2) find :- tells which element belong to which sets

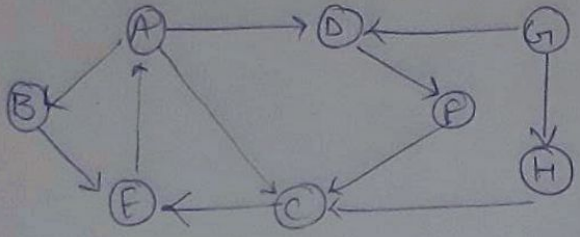
find(1) = S_1 , find(5) = S_2

3) Intersection :- $S_1 \wedge S_2 = \{0\}$, $S_4 \wedge S_5 = \{6\}$.

$S_4 =$ 

$S_5 =$ 

Ques6 Run BFS and DFS on given graph



BFS :- nodes G H F D C E A B

parent ✗ G G G G H C E A

Visited nodes → G H F D E A B

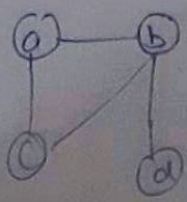
path G → H → C → E → A → B

DFS : nodes processed G G D C E A B

stack G DFH CFH EFH AFH BFH FH

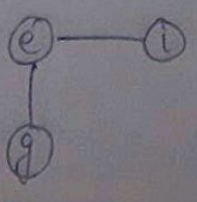
path → G → D → C → E → A → B

Ques7 Find out no. of connected components & vertices in each group. Using disjoint set data structure.

1) 

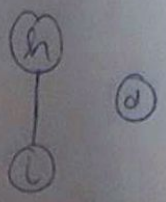
no.(v) = 4

no.(cc) = 1

2) 

no(v) = 3

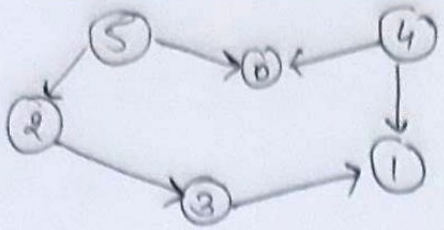
no.(cc) = 1

3) 

no.(v) = 3

no.(cc) = 2 .

Ques 8



stack | 0 | 1 | 3 | 2 | 4 | 5 |

Topological: 5; 4; 2; 3; 1; 0

DFS stack → | 4 | 0 | 1 | 3 | 2 | 5 | Head →

DFS → 5 → 2 → 3 → 1 → 0 → 4

Ques 9

We can use heaps to implement priority queue. It will taken O(logn) time to insert & delete each element in the priority queue. Based on heap structure, priority queue has also 2 types → max & min, priority queue. Some algorithms where we need to use priority queue:

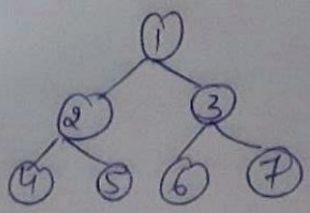
1) Dijkstra's shortest path algo using priority queue:- where graph is sorted in the form of adjacency list or matrix, priority queue can be used minimum efficiently when implementing Dijkstra's algorithm.

ii) Prim's Algorithm:- Priority queue is used to implement prim's to store keys of nodes & extract minimum key node at every step.

iii) Data compression:- Priority queue is used in Huffman's code which is used to compress data.

Ques 10

In minimum Heap, the key is present at the root node must be smaller than among the keys present at all of its collection



In max Heap, the key present at the root node must be greater than among the keys present at all of its collection.